멀티코어 CPU 환경하에서 능률적인 네트워크 관리를 위한 유전알고리즘을 이용한 국부적 RED 조정 기법

A Local Tuning Scheme of RED using Genetic Algorithm for Efficient Network Management in Muti-Core CPU Environment

송 자 영* 최 병 석**
Ja-Young Song Byeong-Seog Choe

요 약

네트워크 장비를 관리함에 있어서 환경에 따른 RED(Random Early Detection) 매개변수에 대한 설정은 쉽지 않은 일이다. 특히 관리자가 환경의 변화에 따라 일정한 서비스율을 유지하고 싶은 경우의 매개변수 설정은 더욱 쉽지 않은 일이다. 본 논문에서는 출력 큐에 멀티 코어 CPU를 탑재한 라우터를 가정하고 라우터의 출력 큐에, RED의 환경에 따른 매개변수의 최적화에 적합한 것으로 알려진, 인공지능의 유전 알고리즘을 직접적으로 도입하여 스스로 부하에 적응하는 AI RED(Artificial Intelligence RED)를 제안한다. AI RED는 FuRED(Fuzzy-Logic-based RED) 보다 단순하고 세밀하며, 실험을 통하여 AI RED가 찾아낸 RED 매개변수는 표준 RED 매개변수보다 환경에 더욱 잘 적응하는 효율적인 서비스를 제공하여 준다는 것을 확인 할 수 있다. RED 매개변수 관리의 자동화는 네트워크 관리의 측면에서 많은 효율성의 향상을 관리자에게 제공하여 줄 수 있다.

ABSTRACT

It is not easy to set RED(Random Early Detection) parameter according to environment in managing Network Device. Especially, it is more difficult to set parameter in the case of maintaining the constant service rate according to the change of environment. In this paper, we hypothesize the router that has Multi-core CPU in output queue and propose AI RED(Artificial Intelligence RED), which directly induces Genetic Algorithm of Artificial Intelligence in the output queue that is appropriate to the optimization of parameter according to RED environment, which is automatically adaptive to workload. As a result, AI RED Is simpler and finer than FuRED(Fuzzy-Logic-based RED), and RED parameter that AI RED searches through simulations is more adaptive to environment than standard RED parameter, providing the effective service. Consequently, the automation of management of RED parameter can provide a manager with the enhancement of efficiency in Network management.

☞ KeyWords : 네트워크 관리(Network Management), 동적 큐 관리(Active Queue Management), 혼잡제어(Congestion Control)

1. 서 론

라우터의 큐 상에서 패킷을 확률에 의해 무작위로, 능동적으로 폐기하여 사전에 전역동기화와 같은 문제를 해결하는 기법을 통틀어 일반적으로 동적 큐 관리(AQM: Active Queue Management)라 한다. AQM의 예로 RED(Random Early Detection)가 있으며 이와 같은 능동적인 라우터 큐에 대한 관리로 인하여 전역 동기화 문제에 대한 해결과 함께 라우터 큐에서 패킷의 지연시간을 작게 유지할 수 있는 장점도 얻을 수 있다.[1] RED는 그 구현의 단순함으로 인하여 매우 널리 사용되고 있으나 다양한 환경에 적용하기에는 많은 어려움을 가지고 있다. 환경에 적합한 RED 매개변수에 대한 적절한 설정을 하기가 쉽지 않기 때문이다. 때문에 환경의 변화에 적응하는 변형된 형태의 RED가 많이 제안되었으며 대표적인 것으로

* 정 회 원 : 동구여자상업고등학교 교사
hpc3341@daum.net
** 정 회 원 : 동국대학교 정보통신공학과 교수
bchoe@dgu.edu
[2009/04/07 투고 - 2009/04/16 심사(2009/06/22 2차) - 2009/07/15 심사완료]

ARED(Adaptive RED)가 있다. ARED의 경우 순간적인 트래픽의 변화에 적응하는 능력을 가지고 있으나 ARED를 운영하는데 필요한 매개변수(α, β)를 설정하는 기준이 없다.[2] 두 값에 대한 결정이 ARED의 성능에 결정적인 영향을 준다고 볼 때 두 값에 대한 설정은 역시 환경의 변화를 인지하는 관리자의 경험에 의해 수동으로 처리해야 하는 문제가 발생한다. 이것은 기타의 RED를 변형한 대부분의 기법에서 발생하는 문제라고 할 수 있다. 이에 따라 네트워크 관리 시에 인공지능 또는 자체 알고리즘을 적용하여 네트워크의 혼잡에 영향을 주는 매개변수들을 조정하는 방법이 모색이 되었다.[3][4] 그러나 기존의 방법들은 단일 라우터에서 국부적으로 운영 가능한 자동화에 대한 관점이 약하다. 인공지능을 이용하여 국부적인 운영의 자동화를 구현한 대표적인 방법으로 FuRED(Fuzzy-Logic-based RED)와 같은 방법들이 있으나, 퍼지(Fuzzy) 알고리즘 기반이며 퍼지알고리즘의 룰베이스 설정등에 유전알고리즘을 부분적으로 적용한 방법들이다.[5] 이는 환경의 변화에 따른 동작의 자동화를 이룰 수 있으나 룰 베이스와 퍼지 셋의 설정 등 초기에 관리자가 설정해야하는 매개변수의 수가 너무나 많아 관리가 힘들고 성능 개선에 대한 불확실성이 크다. 또한 큐 관리의 최적화에 사용되는 매개변수가 max_p 하나이고, max_p의 변화가 퍼지 셋의 개수에 의해 제약받으므로 세밀한 성능의 최적화에 한계가 있다.

본 논문에서는 라우터에서 인공지능을 운영할 수 있는 환경이 마련되어 있다는 가정 하에 라우터의 출력 큐에 인공지능의 유전 알고리즘(Genetic Algorithm)을 직접적으로 도입하여 국부적으로 스스로 환경에 적응하는 AI RED와 그 운영정책을 제안한다.

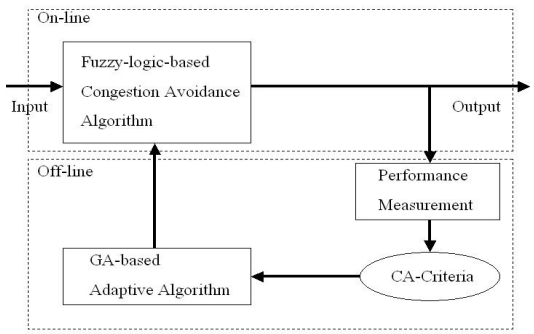
본문을 통하여 네트워크 관리의 측면에서 FuRED보다 AI RED의 운영이 더 단순하고 세밀함을 보이고, 시뮬레이션을 통하여 일반적인 RED 운영방법과 비교함으로써 그 성능을 검증한다.

멀티코어 마이크로프로세서의 발전과 고성능 대용량 메모리의 발전은 지속적인 가격의 하락 속에 컴퓨팅 파워의 향상을 가지고 오고 있으며 이러한 현상을 잘 이용할 수 있는 소프트웨어의 출현은 필연적인 것이라고 할 수 있다. 기존의 연구에서도 퍼지 알고리즘 기반하의 PI 컨트롤러 상에서 룰베이스(Rule Base)의 최적화에 병렬유전알고리즘을 적용한 연구가 존재하며 [6] 멀티코어 CPU는 이의 실현 가능성을 더욱 높혀준다.

AI RED는 단순화된 새로운 운영방법과 구현방법으로 네트워크 관리와 성능의 최적화 측면에서 많은 효율성의 향상을 관리자에게 제공하여 줄 수 있다.

2. 관련연구 : FuRED

TCP/IP 네트워크의 라우터에서 국부적으로 인공지능을 RED에 적용하는 대표적인 예로 FuRED를 들 수 있다.[5] 기존의 인공지능 분야에서는 매개변수의 자동화와 최적화를 구현할 때 많이 사용하는 모델로 퍼지 알고리즘을 기본으로 유전알고리즘이 퍼지 알고리즘을 지원하는 모델(Genetic-Fuzzy System)을 주로 사용한다.[7] 이와 같은 인공지능 모델에 기반하여 PI(Proportional Integral) 컨트롤러 등에 인공지능이 적용되는 모델이 제안되었다.[8][9] 반면에 기존의 RED 큐 동작 모델에 전자의 인공지능 모델을 직접적으로 사용하는 RED가 FuRED이다.



(그림 1) FuRED 동작 구조

FuRED에서는 퍼지 알고리즘의 룰 베이스(Rule Base)와 퍼지 셋(Fuzzy sets)을 최적화 하는데 유전 알고리즘이 부분적으로 이용되며 그 동작 구조는 그림 1과 같다.

그림 1의 FuRED 동작 과정은 두개의 부분으로 나뉘어 지는데 하나는 오프라인 부분이고 다른 하나는 온라인 부분이다. 온라인 부분은 퍼지 알고리즘을 기반으로 RED에 의해 혼잡제어가 동작을 하는 부분이며 오프라인 부분은 성능 측정 시 기준에 적합하지 않을 경우 유전 알고리즘을 이용하여 퍼지 알고리즘의 기준이 되는 룰 베이스와 퍼지 셋 부분의 조절을 담당한다. 그리고 조절된 값에 의해 온라인 부분의 퍼지 알고리즘이 동작하게 된다.

FuRED에서 유전 알고리즘에 의해 최적화되는 값은 표 1과 같은 룰 베이스인데 각각의 셀은 RED의 매개 변수인 max_p 값을 의미한다.

(표 1) FuRED의 퍼지 논리 룰 베이스의 예

△Qlen \ Qlen	VS	S	M	L	VL
VS	VL	VL	VL	L	M
S	VL	VL	L	M	H
M	VL	L	L	H	VH
L	L	L	M	H	VH
VL	M	M	H	VH	VH

FuRED에서 사용되는 퍼지 알고리즘은 입력 매개변수로써 큐 길이인 Qlen과 큐 길이의 변화율인 △Qlen을 가지며 두 매개변수에 따라 출력 매개변수 max_p가 조절된다. 입력 퍼지 매개변수는 각각 다섯 개의 퍼지 셋(VS, S, M, L, VL - V는 Very, S는 Small, M은 Middle, L은 Large)을 가지며 출력 매개변수 역시 다섯 개의 퍼지 셋(VL, L, M, H, VH - V는 Very, L은 Low, M은 Middle, H는 High)을 가진다. 퍼지 알고리즘의 룰은 식 1과 같고 FuRED의 동작은 그림 2를 따른다.

IF Qlen is X and △Qlen is Y then max_p *is Z* (식 1)

with X, Y ∈ {VS, S, M, L, VL}

and Z ∈ {VL, L, M, H, VH}

즉 FuRED는 그림 1과 같은 동작모델의 온라인 하에서 관리자의 손에 의해 세팅된 값으로, 그림 2의 알고리즘에 의해 식(1)을 적용하여 동작하며, 기준에 어긋날 경우 오프라인 모드에서 유전알고리즘에 의해 표 1의 룰 베이스를 최적화하게 된다. FuRED의 장점이라면 장 단기적인 환경 변화에 대처할 수 있는 자동화에 있고 단점이라면 다음과 같은 것을 들 수 있다.

```
For each packet arrival
    Calculate Qlen and △Qlen
    if (min_th <= Qlen < max_th){
        Calculate probability max_p
        with probability max_p
        Drop the arriving packet}
    else if (max_th >= Qlen)
        Drop the arriving packet
```

(그림 2) FuRED 동작 알고리즘

첫째, 초기에 관리자가 수동으로 처리해야 하고 설정해주어야 하는 매개변수(퍼지 셋들과 룰 베이스와 기타 관련된 변수들)가 너무 많고 매개변수가 너무 많은 만큼 매개변수의 세팅에 의해 발생할 수 있는 성능의 개선에 대한 확률적 불확실성이 너무 크며 관리자에게도 불편을 안겨 준다.

둘째, FuRED의 동작과 룰 베이스의 최적화에 사용되는 RED의 매개변수가 max_p 하나이고 max_p의 변화가 퍼지 셋에 의해 5개의 단계로만 구분되어 동작하게 되어 있으므로 성능의 세밀한 최적화에 한계가 있다. 참고로 RED에서 사용하는 min_{th}와 max_{th}와 같은 매개변수도 RED 성능의 최적화에 도움을 준다.

셋째, 부하에 적응하기 위해 룰 베이스의 25개의 max_p 값에 대한 최적화가 이루어 져야 하는데 너무 많은 매개변수에 대한 최적화가 이루어 져야 하므로 오프라인 쪽에서 시뮬레이션 시 매우 큰 컴퓨팅 파워가 요구되며 비효율적이다.

3. AI RED

AI RED는 단기적인 트래픽의 변화가 아니라 중장기적인 환경의 변화에 대처하는 것을 목표로 한다. 일반적으로 ARED와 같이 단기적인 폭주 트래픽에 대처하기 위해 개선된 RED등이 기존의 RED에 비하여 그리 의미를 가지지 않는 이유는 큐와 기존의 RED에서 상당부분 흡수할 수 있는 일시적인 폭주 트래픽에 대한 반응보다는 그런 폭주 트래픽이 지속적으로 발생하는 환경에 대한 반응이 네트워크 관리자의 입장에서 더 중요하기 때문이다.

AI RED의 전체 동작 과정의 개요는 다음과 같다. 네트워크 관리자는 라우터의 특정 출력 포트의 운영 정책을 선택한다.(3.2 참조) 완전 자동화 정책을 선택하였을 경우 6개 (X, Y, EDR:Expected Drop Rate, ECU:Expected CPU Utilization Rate, Standard Fitness, Period)의 AI RED 매개변수를 설정하고 동작시키면, 혼잡이 발생할 경우 주기적으로 독자적인 동작과정(3.3, 4.1 참조)에 기반한 멀티코어 CPU 환경의 유전알고리즘이 동작하여 RED 매개변수를 설정한 후, 혼잡상태가 종료될 시점까지 혼잡모드에서 동작하고 혼잡 상태가 정리되면 보통모드로 동작을 한다. 그리고 반자동화 정책을 선택하였을 경우 관리자가 3개(EDR, ECU, Standard Fitness)의 매개변수를 이용하여 주기적으로 최적화된 RED 매개변수를 설정한다.

3.1 적용조건

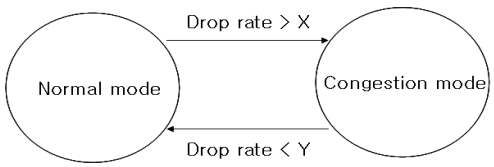
AI RED의 적용을 위해서는 하나의 적용조건이 필요하다. 즉 각 출력 포트에 멀티코어 CPU와 대용량의 메모리가 탑재되어 있다는 적용조건이다.

지난 세월 동안 기존 CPU 구조의 진화는 단일 스레드(Single-thread)의 성능 향상에 치우쳐 있었던 것이 현실이었다. 그러나 전력소모량의 증가와 그에 따른 열 문제로 인해 이후 CPU 구조의 진화 패러다임을 멀티스레드(Multi-thread)의 성능 향상에 초점을 맞추게 된다.[10] 2008년에 듀얼코어 CPU가 이미 보편화 되었으며 2009년에는 쿼드코어 CPU가 널리 보급될 것으로 보이고 그 이후에는 8~32코어의 시대가 열릴 것으로 예상된다. 게다가 하나의 코어가 여러 개의 스레드를 동시에 지원한다면 논리적으로 더욱더 많은 코어가 존재하는 효과를 가진다. (예: Sun의 UltraSPARC t2의 경우 8코어 64스레드, Intel의 i7 4코어 8스레드) 최근의 멀티코어 CPU는 각 코어에서의 명령어 수준 병렬성(ILP: Instruction Level Parallelism)이나 전력의 효율성도 기존의 단일코어 CPU보다 좋기 때문에 가격의 하락만 충분히 이루어진다면 출력 포트에서 강력한 컴퓨팅 파워를 제공해 줄 수 있을 것이다. 이 하드웨어적 전제조건을 기반으로 AI RED가 동작하게 된다. 논의의 단순화를 위해 출력 포트에서 큐잉이 발생하는 것으로 가정하며, 라우터는 ECN-noncapable 라우터로 가정한다. 또한 출력 포트의 RED 큐에서는 스케쥴링 알고리즘은 FCFS를 사용하며 라우터에서 일정한 트래픽 양으로 조정하여 약속한 양만큼 전송하는 트래픽 쉐이핑(Traffic-shaping)은 사용하지 않는 것으로 가정한다.

3.2 운영정책

AI RED의 운영 정책은 두 가지의 정책을 사용할 수 있다. 첫째, 완전 자동화 정책(Full-automation policy)이다. 완전자동화 정책은 관리자가 설정해 놓은 서비스 변수를 기반으로 스스로 알아서 작동하는 정책을 의미한다. 즉 항상 주어진 매개변수를 기반으로 환경에 맞추어 RED 매개변수를 최대한 최적화 시켜 동작하는 방법이다. 관리자는 라우터를 설정할 때에 자신이 완전자동화 정책을 사용한다고 선택할 경우 그림 3에서와 같이 두 가

지 모드 중 하나를 선택할 수 있다. 하나는 보통 모드(Normal mode)이고 다른 하나는 혼잡모드(Congestion mode)이다. 보통모드는 표 2의 Floyd가 제안한 RED 매개변수 값 [11] 으로 동작하는 모드를 의미하고 혼잡모드는 AI RED 알고리즘이 동작하여 RED 매개변수 값을 세팅하고 그 값으로 실제 RED 큐가 동작하는 모드를 의미한다.



(그림 3) 완전 자동화 AI RED 운영 정책 상태 천이도

보통모드에서 폐기율이 혼잡모드로 진입하는 기준인 X보다 커지면 AI RED 동작모드로 진입하게 된다. 이후 AI RED에서 찾아진 매개변수로 동작하다가 폐기율이 혼잡모드에서 보통 모드로 진입하게 되는 기준 Y보다 작아지면 표 2의 매개변수 값으로 다시 설정하여 동작하는 보통 모드로 동작하게 된다. 이 폐기율에 관한 통계적 계산은 라우터에서 로그 기록을 이용하여 관리자의 주기(Period) 설정에 따라 주기적으로(예를 들면 1시간, 1일, 또는 1주일) 실행 한다.

(표 2) Floyd의 RED 매개변수 권고치

max_p	0.1
min_{th}	5 packet
max_{th}	15 packet
w_q	0.002
Queue Length	25 packet

관리자는 X와 Y값(예: X=0.135, Y=0.03)과 혼잡모드에서 동작할 서비스 변수 값(예: ECU=0.6, EDU=0.05, Standard Fitness=0.6, Period=1hour)을 설정해 놓은 후 라우터를 동작시킨다. 이후에는 자동으로 RED 매개변수 값이 갱신되어 동작하게 된다. 완전 자동화 정책에서 큐는 관리자가 설정한 혼잡상황(예: X>0.135)에서 주어진 부하에 최대한 적응하려 노력하게 된다. X와 Y 값의 차가 너무 적게 되면 두 모드 사이에서 빈번한 스위칭이 발생하므로 성능의 저하가 발생 할 수 있으므로 X와 Y값은 적어도 0.1 이상의 차이를 두고 설정하는 것이 적절하다.

둘째, 반자동화 정책(Semi-automation policy)이다. 반자동화 정책은 네트워크 환경의 변화에 따라 관리자가 인터넷을 이용하여 원격으로 라우터의 출력 큐에 대한 로그 정보 확인하고, 로그 정보에서 출력 큐 CPU의 이용율과 패킷의 폐기율 등을 확인한 후, 그 확인된 결과를 기반으로 관리자가 라우터의 서비스 변수를 설정 하는 방법이다. 서비스 변수가 설정이 되면 그 값을 최대한 만족하도록 AI RED가 작동하여 자동으로 RED 매개변수를 설정하게 된다. 이 정책은 완전 자동화 정책을 보조한다.

그림 4는 AI RED의 운영정책에 대한 동작 알고리즘이다.

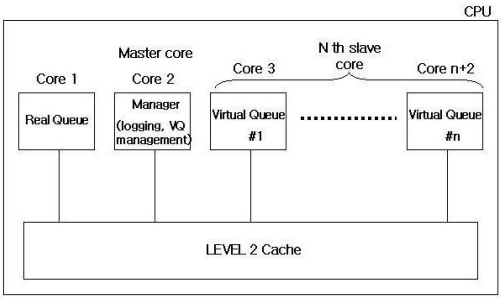
```
Full-Automation policy :
Default_parameter set  //Manually, Floyd parameter
AI_RED_parameter set //Manually, AI RED parameter
Current_mode=0
Per every interval of period
//Mode 1 is congestion, 0 is normal
if(Drop_rate > X){
     Current_mode=1
     RED_parameter=AI_RED_START(AI_RED_parameter)
     Set_RED(RED_parameter)}
else if(Drop_rate < Y and Current_mode==1){
       Current_mode=0
       Set_RED(Default_parameter)}
Semi-Automation policy :
AI_RED_parameter set //Manually, AI RED parameter
RED_parameter=AI_RED_START(AI_RED_parameter)
Set_RED(RED_parameter) //Manual Setting
```

(그림 4) AI RED 운영정책 동작 알고리즘

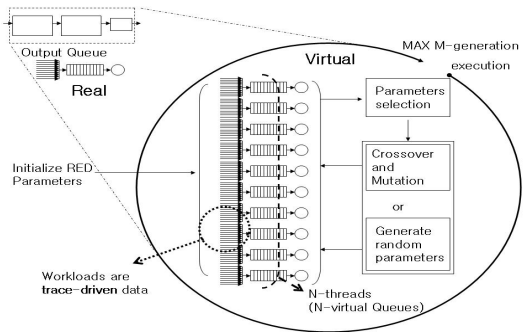
3.3 설계

AI RED에서는 큰 그레인 병렬성(Coarse-grain parallelism)을 이용한다. 큰 그레인을 사용하면 미

세 그레인 병렬성(Fine-grain parallelism)에 비해 각 스레드 간에 통신의 오버헤드를 최소화 할 수 있기 때문에 하나의 스레드는 하나의 코어에 대응시킨다. 통신의 오버헤드는 민스키의 모순성(Minsky's conjecture)에서도 알 수 있듯 병렬처리의 성능 향상을 막는 주원인이기 때문이다.[12] 단 각 코어의 효율성은 상대적으로 떨어질 수 있다.



(그림 5) AI RED의 코어 개념도



(그림 6) AI RED의 동작 개념도

AI RED는 그림 5, 그림 6과 같은 개념을 기반으로 동작 한다. 출력 큐에 코어의 개수가 (N+2)개인 멀티코어 CPU가 존재하며, (N+2)개인 멀티코어에서 두 개의 코어 중 하나는 실제 큐에 대한 처리를 담당하고 다른 하나의 코어는 입력 트래픽에 대한 로그 파일 작성과 가상 큐에 대한 랜덤 넘버 작성, 그리고 AI RED 동작 시 마스터(Master) 스레드가 위치하는 관리 코어의 역할을 담당한다.

AI RED의 운영 정책에 의해 실제 큐의 RED 값을 조절해야 한다는 결정이 내려지면, 관리자에 의해 주어진 AI RED 매개변수의 서비스 변수인 EDR, ECU를 기준으로 그 변수에 적합한 RED 매개변수 값을 찾기 위해 나머지 N개의 코어들은 각각에 할당된 슬레이브(Slave) 스레드 안의 가상(Virtual)의 큐에서 로그 파일을 이용하여 시뮬레이션을 한다.

여기서 RED 매개변수 값(본 논문에서는 max_p, min_{th}, max_{th})의 모임은 개체(Individual)라 하며 max_p, min_{th}, max_{th} 각각은 유전자(Gene)라 한다. 그리고 max_p와 같은 값에 존재하는 한 비트를 우리는 염색체(Chromosome)라 한다.

```
Function AI_RED_START(AI_RED_parameter)
Loop=0 // Counter of execution
Initialization of each of N_individuals in N_threads
                                using Random number()
N_threads simulation() using AI_RED_parameter
Fitness=MIN(Fitness calculation of each of N_individuals())
while(Fitness>Standard fitness && Loop<=M_generation-1){
   //Genetic Operation - Reproduction
   Selection(); Crossover(); Mutation(); with N_individuals
   Initialization of reminder individual of unselected
                                threads using Random number()
   N_threads simulation() using AI_RED_parameter
   Fitness calculation of each of N_individuals()
   Loop++
}
RED_parameter=Select a Individual of minimum fitness
                                                value()
Return RED_parameter
END Function
```

(그림 7) AI RED 동작 알고리즘

AI RED는 그림 7과 같은 알고리즘에 따라 최적의 매개변수를 형성하게 되는데 각 스레드는 랜덤 넘버에 의해 각각의 RED의 초기 값을 설정 후 그 값에 의해 시뮬레이션을 하게 되며 각각의 스레드는 가상 큐에서 나온 결과와 AI RED 매개변수를 이용하여 적합도(Fitness)를 구한다. 마스터 스레드는 그 적합도에 순위를 매겨 우성 유전자를 가진 2개의 개체들을 선택(Selection)한 후 우성유전자를 가지고 교배(Crossover)와 돌연변이

(Mutation)를 거쳐 식 2만큼의 새로운 유전자를 가진 개체들을 만들어 낸다. 단 기존의 선택된 유전자는 보존한다. 그리고 나머지 (최대개체개수 - 식 2) 만큼의 개체는 랜덤하게 새로운 RED 유전자를 받아들여 다음 세대를 위한 유전자 배치를 완료한다.

$$(N/2-2)\times 2,\ N\text{: The Number of Max object} \qquad \text{(식 2)}$$

즉, 우성유전자를 가지는 스레드와 새로운 개체 이외의 기존의 열성유전자를 가지는 스레드는 모두 새로운 랜덤 값으로 설정한다. 이후 다시 시뮬레이션을 시작하며 이 과정이 M번의 세대를 거치거나 아니면 M세대 안에 기준 적합도(Standard fitness)보다 작은 적합도를 가지는 적합한 유전자를 가지는 개체를 찾게 되면 실제 큐에 그 값을 적용 시킨다.

M세대 동안 실행되는, AI RED 루프에서 실행 종료의 조건이 되는, 기준 적합도는 관리자가 입력하는 서비스 변수 중 하나이며, 적합도는 EDR 또는 ECU와 같은 관리자가 입력한 서비스 변수들과 한 AI RED 개체의 실험결과에 의해 만들어진 폐기율과 이용율에 대한 분산의 합으로 정의할 수 있다. AI RED의 구현에서는 식 3과 같이 적용한다. 식 3에서 β, γ 는 조정 변수로써 각각 β =100, γ=100의 값을 가진다.

$$Fitness = (DR-EDR)^2\times\beta+(CU-ECU)^2\times\gamma \qquad \text{(식 3)}$$

DR: Drop Rate, EDR: Expected Drop Rate

CU: CPU Utilization Rate, ECU: Expected CPU Utilization Rate

일반적으로 기준 적합도는 조금 무리한 기준을 제시하는 것이 우리가 좋은 유전자를 얻을 수 있는 방법이라 하겠다. 예를 들어 적합도가 기준적합도(예: 0.3) 미만이어야 우리가 만족하는 값이 나올 것이라 관리자가 판단하면 그것이 조금 무리한 기준이라 하더라도, AI RED는 기준에 만족하지는 못하지만 가장 근사치의 값을 주어진 세대 안에 찾아낸다.

일반적인 싱글코어 CPU에서 유전 알고리즘 계산에 걸리는 시간(P)은 식 4의 값을 가지게 된다.

$$P=(K\times N+X)\times M \qquad \text{(식 4)}$$

K : the time of a thread calculation

N : the number of individuals (or objects)

M : the number of generations

X : the time of a reproduction

AI RED에서 사용하는 N개의 멀티코어 환경 하에서의 P는 한 개의 코어가 한 개의 유전 개체에 대한 스레드를 가지게 되므로 식 5의 값을 가지게 된다.

$$P=(K+X)\times M+A \qquad \text{(식 5)}$$

K : the time of a thread calculation

M : the number of generations

A : inter-thread communication time

X : the time of a reproduction

식 5에서 스레드 간 통신 시간(A)을 최소화 한다면 당연히 N개의 멀티 코어 환경 하에서 성능의 개선은 이론 상 N배에 근접하게 되며 이것은 AI RED가 작업부하에 적응하는 속도를 높여 줄 수 있다. AI RED에서는 큰 그레인을 사용하여 마스터-슬레이브 간의 스레드 통신에 필요한 내용이 유전정보의 설정이나 적합도의 보고 등과 같이 양이 매우 적으며, 거의 동시에 그와 같은 사건이 발생하고 처리된다. 때문에 식 4에서 증가하는 스레드의 수에 따른 계산시간의 증가를 감안하면 식 5에서의 스레드 간 통신 시간의 증가는 미미한 수준으로 볼 수 있다.

AI RED에서 사용되는 유전알고리즘의 정책은 선택 연산자의 경우 순위 선택(Ranking selection)과 엘리트 보존 선택(Elitist preserving selection)을 사용하며, 교배 연산자는 일점교배(One-point crossover)를 사용한다. 돌연변이 연산자의 경우

돌연변이율(Mutation probability)은 조금 더 돌연변이를 많이 만들기 위해 일반적인 기준(5%)보다 높은 20% 정도를 사용한다.

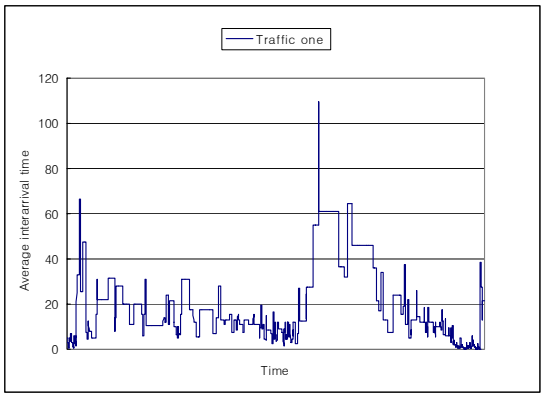
4. 시뮬레이션

본 시뮬레이션에서는 일반적인 RED와 AI RED 기법의 성능비교를 한다. 기존의 국부적인 완전 자동화 방법인 FuRED와 새롭게 제안한 AI RED의 경우 성능의 비교가 아닌 네트워크 관리시 운영의 편의성과 매개변수 조정에 대한 세밀성의 비교이며 이는 관련연구와 결론에 언급되어있다.

4.1 구현

본 논문에서 작업부하들은 TCP의 느린 시작(Slow start) 단계를 지나 혼잡 회피(Congestion avoidance)와 빠른 재전송(Fast retransmit) 그리고 빠른 복구(Fast recovery) 단계를 반복하고 있는 것으로 가정한다. 사용자 간의 상관관계가 낮은 경우를 가정하여 작업부하에 사용되는 트래픽은 포아송(Poisson) 분포를 따르고 라우터의 큐를 거치는 흐름들의 패킷도착이 연결들 사이에 비교적 균일하게 분포되어 있는 경우를 가정한다. Web 트래픽만을 분석한 결과를 보면 사용자의 상관관계가 높을 경우 사용자들의 트래픽이 버스트(Burst)한 특성을 보이므로 트래픽이 자기 유사(Self-Similar)한 특징을 가지는 것으로 알려져 있으나 사용자의 상관관계가 낮을 경우 사용자들이 발생시키는 트래픽이 분산되어 트래픽이 포아송 분포를 보이는 것으로 알려져 있다.[13] 그리고 패킷의 폐기에 따라 λ(Inter-arrival rate, 도착간격율)에 변화를 주어 TCP 트래픽을 근사모델링 하였다. 그림 8은 TCP 근사 알고리즘에 의해 표 2의 매개변수를 사용하는 RED 시뮬레이션에서 한 개의 작업부하 소스에서의 평균도착간격시간의 부분적인 변화량을 보여준다.

또한 패킷 크기는 고정된 것으로 가정한다. 현재 네트워크 엑세스 계층의 경우 사실상 이더넷이 표준이기 때문이기도 하고 일반적으로 발신지에서 목적지 까지 Path MTU로 전송하는 것이 불필요한 분할을 발생시키지 않아 바람직하며 IPv6는 호스트가 이를 발견할 수 있는 표준 메커니즘을 제공한다.

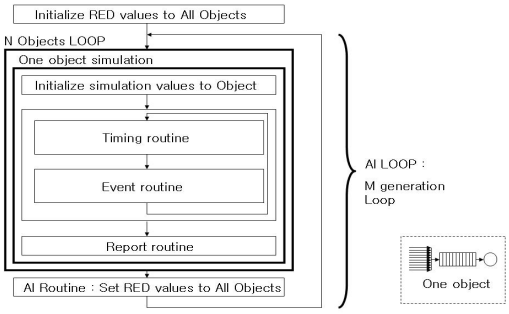


(그림 8) 평균도착간격시간의 변화

RED에 인공지능 개념을 추가하여 만든 AI RED에 대한 시뮬레이터는 일차적인 목적이 기존의 RED와 비교하여 혼잡상태에서의 성능을 알아보는 것에 있다. 실제 동작은 그림 8과 같이 동작하도록 구성하였다.

유전알고리즘에서 인공지능 시뮬레이션은 일반적인 시뮬레이션보다 몇 배에서 몇 십 배 많은 연산을 필요로 하며 기존의 네트워크 시뮬레이션 도구(NS-2, OPNET)들은 네트워크 시뮬레이션에 특화되어 매우 프로그램의 크기가 크고 무거우며 유저의 인터페이스 부분에 사용되는 언어 등이 정적환경에 최적화(예: NS-2의 OTCL) 되어 있기 때문에 인공지능 시뮬레이션에 적용 시 적절하지 못하다. 때문에 큐잉 이론을 기반으로 하여 C언어로 멀티스레드 환경을 모델링 한 후 프로그래밍하고 시뮬레이션 하였다. 본 시뮬레이션은 하나의 노드에 AI RED를 적용하는 방법을 사용하였다. 하나의 객체(Object)는 그림 9에서 볼 수 있듯 하나의 RED 큐를 의미하는 것으로 RED 큐에 대한 RED 변수와 시뮬레이션 변수들로 구성되어 있으

며 하나의 객체에 대한 시뮬레이션이 완료되면 초기화 하여 최대객체개수 N(AI RED에서 코어의 수에 따라 달라지는 병렬 수행의 정도)만큼 반복하여 시뮬레이션을 하게 하였다. 이는 본 논문의 시뮬레이션을 하기 위한 환경이 단일 스레드 상에서 이루어지기 때문에 하나의 세대를 N번 반복하여 처리한 것이다. 본 실험에서 사용하는 RED 개체의 유전자는 3가지로 max_{th}, min_{th}, max_p 이다.



(그림 9) AI RED 시뮬레이터 동작 과정

본 시뮬레이션은 기본적으로 원하는 적합도가 나올 때 까지 시뮬레이션을 실행하며, 예외적으로 20세대 또는 100세대까지 원하는 결과가 나오지 않을 경우 가장 근사치의 결과 값을 가지는 개체의 RED 유전자를 얻도록 하였다. 각 개체의 유전자의 값을 이용하여 시뮬레이션 후 만들어진 한 개체에 대한 적합도 평가는 식 3을 따른다. 식 3에 의해 계산된 적합도가 0.6 보다 작은 경우 우리가 원하는 적합한 값을 찾은 것으로 판단하고 그 RED 개체의 유전자를 실제 RED 큐에 적용한다. 기준적합도 0.6은 관리자가 어느 정도의 분산을 허용하는 가에 대한 문제이다.

본 시뮬레이션에서 사용된 인공지능에 관련된 함수 중 선택 함수의 경우는 개체의 적합도들을 정렬하여 선택하고 선택된 개체를 보존하는 것만으로도 쉽게 구현 할 수 있다. 교배 함수와 돌연변이 함수의 경우 min_{th}와 max_{th}와 같은 정수 연산에서는 C언어에서 제공하는 비트연산에 의하여 랜덤하게 선택된 한 비트를 기준으로 두 입력의 값을 교환한다던지, 그 비트 값을 보수연산에 의해 치환하는 방식으로도 충분하지만, 실수 연산의 경우에는 표현방법자체가 부동소수점 표현방법을 사용하므로 복잡하게 된다. 실수 연산에서는 부동소수점 표현방식으로 표현된 실수의 값을 비트스트링으로 변환시킨 후, 그것에 교배 함수와 돌연변이 함수를 적용하고 다시 부동소수점 표현방법으로 변환하는 과정을 거쳐야 하는데 이 과정은 매우 복잡하며 많은 시간을 낭비하게 된다.

우리가 원하는 것은 유전자의 비트를 변화시킴에 따라 유전자의 값이 변화를 하거나 그 지정된 비트의 값을 기준으로 두 개의 유전자가 서로 교배를 하는 것이므로 실수 자체에서 바로 그 연산을 실행하게 하면 복잡성을 줄일 수 있고 속도에서도 이득을 볼 수 있다. 때문에 본 시뮬레이션에서는 실수를 가지고 직접적으로 교배와 돌연변이를 적용하는 방법을 사용하였다.[14] 다만 참고 문헌에 제안된 알고리즘과 문헌의 코드에서 부분적으로 잘못된 부분이 발견되어 수정하였다.

(표 3) 실수에서 비트와 값의 표현의 인코딩 예제

순서	비트	값
0	000	0
1	001	0.125
2	010	0.25
3	011	0.375
4	100	0.5
5	101	0.625
6	110	0.75
7	111	0.875

실수 데이터 타입에 대한 한 비트 돌연변이 알고리즘은 간단한 예를 들어 설명하여 보자. 본 논문의 RED에서 사용하는 max_p의 경우 0.0~1.0의 범위의 숫자를 가지게 된다. 이 범위의 숫자를 3비트(8개의 숫자)로 인코딩(encoding)하여 표현하면 [표 3]과 같이 표현 할 수 있을 것이다.

[표 3]의 2번째 행을 통하여 낮은 비트의 한 비트 값(V)은 0.125의 값을 가짐을 알 수 있다. 그

것을 일반적으로 표현하면 식 6와 같다.

$$V = \frac{Dmax - Dmin}{2^k} \quad \text{(식 6)}$$

Dmax : the largest value in the ranges
Dmin : the smallest value in the ranges
k : the number of bits in the representation

식 6에 따라 표 2의 V는 0.125=(1.0-0.0)/2^3 와 같이 계산된다. 또한 표 2에서 변수 X의 한 비트를 보수 연산하기 위해서는, X의 n 비트가 1이면 $2^n \times V$를 빼면 되고 X의 n비트가 0이면 $2^n \times V$를 더하면 된다는 것을 알 수 있다. n비트의 값이 0인가 1인가를 판단하는 식은 식 7과 같이 구할 수 있다.

$$Bit_n = \left\lfloor \frac{X - Dmin}{2^n \times V} \right\rfloor, Modulo\,2 \quad \text{(식 7)}$$

실수의 교배 알고리즘은 기존의 식을 응용하여 구현할 수 있다. 예를 들어 위의 변환에 의해 만들어진 10비트로 인코딩된 실수에 대하여 10비트 중 n비트 이상(9~n비트)의 비트들로 표시된 영역의 값은 식 8과 같다.

$$X_h = \left\lfloor \frac{X - Dmin}{2^n \times V} \right\rfloor \times 2^n \times V \quad \text{(식 8)}$$

n비트 미만의 값은 $X_l = X - X_h$에 의하여 구 할 수 있다. 그리하여 두 실수 데이터의 교배 알고리즘은 $Y = Y_h + X_l$, $X = X_h + Y_l$ 이라는 공식에 의해 구할 수 있다.

위 알고리즘은 변수 X가 V의 배수 값 이어야만 정확하게 동작을 한다. 때문에 max_p의 값은 k비트의 크기를 늘려 V를 조금 더 세분화 하여 V의 배수로 동작하도록 만들었다. 참고로 시뮬레이션에서 사용한 k비트의 값은 12이며 때문에 V는 0.000244140625=(1.0-0.0)/2^{12} 의 값을 가진다.

4.2 실험결과

앞 절에서 논의한 결과에 따라, 다음과 같은 사용자 설정 값(X=0.135, Y=0.03, EDR=0.05, ECU=0.60, Standard Fitness=0.60, Period=1hour)을 이용하여, 개체 당 50만개의 작업부하와 20세대의 AI RED 알고리즘을 통해 얻은 RED 개체의 유전자와 데이터는 표 4와 같다.

(표 4) AI RED 매개변수 산출 결과(20세대)

AI Object per Generation / Item	5	10	20	40
	A	B	C	D
fitness	0.694031	0.686127	0.555189	0.625465
minth	11	11	10	12
maxth	19	24	14	15
maxp	0.612793	0.867188	0.013428	0.480469
generation for fitness to find	16	6	14	16
termination	20	20	14	20

표 4에 따르면 20세대로는 세대 당 개체가 충분히 크지 않으면 우리가 원하는 적합도를 얻지 못하고 있음을 알 수 있다. 예외적으로 세대 당 개체의 수가 20인 경우, 즉 20개의 스레드가 동시에 실행이 되어 값을 찾아내면, 14번의 세대 만에 원하는 적합도를 얻어 낼 수 있음을 알 수 있다. 이것은 확률상의 문제로 다른 세 경우의 적합도를 보았을 때는 일반적으로 세대 당 개체의 수가 작을 경우 작은 세대 동안 우리가 원하는 적합도를 찾을 확률이 낮다는 유전 알고리즘의 일반론에 부합하는 결론이라 할 수 있다.

확인을 위해 추가로 세대를 더 늘려 100 세대의 AI RED 알고리즘을 통해 얻은 RED 개체의 유전자와 데이터는 표 5와 같다. 개체의 수가 5, 10, 40 개의 경우만을 놓고 볼 때 개체의 수가 많을 경우 역시 빨리 원하는 기준 적합도에 가까워짐을 확인 할 수 있다.

(표 5) AI RED 매개변수 산출 결과(100세대)

AI Object per Generation / Item	5	10	20	40
	E	F	G	H
fitness	0.684994	0.576042	0.555189	0.593067
minth	11	10	10	13
maxth	19	14	14	15
maxp	0.611816	0.011719	0.013428	0.947266
generation for fitness to find	55	58	14	38
termination	100	58	14	38

E의 경우는 100세대 안에서 값을 찾지 못하였으며 추가로 더 많은 세대의 연산을 필요로 하고 있다. G의 경우 확률상의 예외로 볼 수 있기 때문에 표 5의 실험 결과 또한 개체의 수가 많을 경우 유전 알고리즘은 짧은 세대 안에 원하는 적합도를 얻을 확률이 높다는 일반론에 부합한다.

(표 6) RED와 AI RED 실험 결과 비교

	RED Queue :standard parameter	AI RED		
		F	G	H
Average delay in queue	5.642	7.048	7.042	6.845
Average number in queue	4.874	7.019	7.115	6.801
CPU utilization	0.518	0.598	0.606	0.596
Drop percent	0.139	0.126	0.125	0.127
Time of a simulation ended	1157453.234	1004107.331	989882.314	1006461.648

표 6은 완전 자동화 운영방법에서 동일한 작업부하를 준 경우 보통 모드에 해당하는 표준 매개변수를 사용한 RED와 혼잡모드에 해당하는 F, G, H 매개변수를 RED에 적용한 시뮬레이션 결과를 나타내고 있다.

표 6에서 AI RED가 개체 수에 따라 적합도에 어울리는 다양한 RED 매개변수 유전자를 얻어내고 있으며 그 값들은 관리자가 원하는 목표에 근사하도록 조정되어 있음을 확인 할 수 있다.

AI RED는 작업부하에 적응하여 최대한 적합한 값을 찾아내었으며 표준 RED와 가장 좋은 적합도를 가지는 G를 비교할 경우 본래 관리자가 달성하려 했던 CPU 이용율과 패킷의 폐기율에서 많은 효율의 향상을 가지고 왔음을 알 수 있다. F, G, H의 경우 평균 큐 크기에서 확인할 수 있듯이 큐를 충분히 활용하고 있으며 이로 인하여 큐를 조금 더 효율적으로 이용하고 있음을 알 수 있다. G의 자료를 기준으로 CPU 이용율은 14.5% 향상되었으며, 패킷의 폐기율은 11.2% 더 낮아져 있다. F는 F의 자료를 기준으로 CPU 이용율은 13.8% 향상되었으며, 패킷의 폐기율은 10.3% 더 낮아져 있다. H는 H의 자료를 기준으로 CPU 이용율은 13.1% 향상되었으며, 패킷의 폐기율은 9.4% 더 낮아져 있다. F, G, H에서 AI RED를 이용하여 CPU의 이용율을 향상시켰기에 전반적으로 시뮬레이션의 시간이 줄어들어 있음을 알 수 있다.(G의 기준으로 16.9%의 시간 절약) 이를 실제 통신에 투영하였을 경우 전송 흐름상의 호스트 입장에서는 더 짧은 시간 안에 많은 양을 효율적으로 전송했음을 의미한다. 어떠한 경우이던지 AI RED가 찾아낸 RED 매개변수는 표준 RED 매개변수보다 관리자가 원하는 더 효율적인 서비스를 제공하여 준다는 것을 확인 할 수 있다.

더 중요한 것으로 표 6을 통하여 관리자가 주어진 작업부하 환경에서 확인 할 수 있는 것은 주어진 작업부하의 특성으로 인하여 CPU의 이용율을 일정 수준으로 유지하면서 폐기율을 자신의 기준에 맞도록 낮추는 데는 한계가 있다는 점일 것이다. 이 경우 관리자는 폐기율이 자신이 생각하고 있는 관리의 기준에 적합하지 않다고 판단될 경우 추후 네트워크에 대한 재설계의 근거 자료 등으로 활용 할 수 있을 것이다.

5. 결 론

본 논문에서는 라우터의 출력 큐에 멀티코어 CPU를 사용하였을 경우를 가정하고 그 환경 하

에서 인공지능 분야의 유전알고리즘을 적용하여 네트워크 관리의 편의성을 향상시킬 수 있는 AI RED를 제안하였다. 시뮬레이션을 통하여 일반적으로 더 많은 멀티코어 사용 시 더 빠른 세대 동안에 원하는 RED 매개변수를 얻을 수 있음을 확인하였다. 또한, 작업부하에 따라 AI RED가 표 2의 표준 RED 매개변수보다 더 효율이 좋은 RED 매개변수를 얻는 것을 알 수 있었고, 세밀하게 관리자가 원하는 적절한 서비스 변수에 대한 RED 매개변수의 근사 값을 자동으로 찾아냄도 확인할 수 있었다.

본 논문에서 제안한 유전 알고리즘을 적용한 AI RED의 가장 큰 장점은 바로 이 자동화를 간편하게 구현하고 있다는 것에 있다. 기존의 ARED와 같은 알고리즘은 많은 경우 매개변수에 대한 수동적 설정이 필요한 반면에 AI RED의 경우는 완전 자동화 운영 방법의 경우 초기 서비스 변수 값만 설정을 해주면 혼잡모드와 보통모드를 반복하며 이후 추가적인 설정이 필요치 않은 방법을 제공한다.

또한 네트워크 관리의 편의성 측면에서 FuRED의 경우는 △Qlen, Qlen, Max_p의 퍼지셋 15개와 룰베이스의 초기값을 관리자가 고려하여 초기 설정해야 하는 것과 다르게, AI RED는 단순화된 자동화 모델을 제공하여 관리자가 고려해야 하는 매개변수의 개수를 최소화(완전자동화 모드의 경우 6개 : X, Y, EDR, ECU, Standard Fitness, Period) 한다. 그리고 max_p 하나만을 5단계로만 증감 시키는 FuRED와 달리 min_{th}, max_{th}(1씩 증감), max_p(최소 0.000244140625의 값으로 증감) 매개변수를 이용한 세밀한 제어를 통해 중장기적인 환경의 변화에 대한 세밀한 적응성을 보장하며, 멀티코어 CPU의 이용을 가정하여 유전알고리즘 활용에 병렬성을 이용하므로 최적화에 들어가는 시간을 단축할 수 있다.

또한 AI RED의 경우 여타의 조정 방법들처럼 네트워크의 관리자에게 제공할 수 있는 자동화된 도구로써의 의미도 가진다.

참 고 문 헌

[1] Sally Floyd and Van Jacobson, "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking V.1 N.4, August, 1993.

[2] W. Feng, "A Self-Configuring RED Gateway", IEEE INFOCOM99, March, 1999.

[3] Mario Barbera, "A Simulation Tool for Tuning IP Network Parameters Based on Fluid-Flow Models and Parallel Genetic Algorithms", IEEE Globecom, 2005.

[4] Tao Ye, "Adaptive Tuning of RED Using On-line Simulation", IEEE Globecom, 2002.

[5] M. Gan, E. Dorner, J. Schiller, "Applying computational intelligence for congestion avoidance of high-speed networks", Distributed Computing Systems Proceedings 7th IEEE Workshop, 1999.

[6] G. Di Fatta, G. Lo Re, A. Urso, "Parallel genetic algorithms for the tuning of a fuzzy AQM controller", LNCS Proc. of ICCSA 2003, May, 2003.

[7] F. Hoffmann, "Evolutionary algorithms for fuzzy control system design", Proceedings of the IEEE Volume 89 Issue 9, Sep, 2001.

[8] G. Di Fatta, F. Hoffmann, G. Lo Re, A. Urso, "A Genetic Algorithm for the Design of a Fuzzy Controller for Active Queue Management", Systems Man and Cybernetics Part C: Applications and Reviews IEEE Transactions on Volume 33 Issue 3, Aug, 2003.

[9] Piero P. Bonissone, "Genetic Algorithms for Automated Tuning of Fuzzy Controllers: A Transportation Application" ,Fifth IEEE International Conference on Fuzzy Systems, Sep, 1996.

[10] Jon Stokes, "Inside machine", No starch press,

2007.

[11] Sally Floyd, "RED: Discussions of setting Parameter",http://www.icir.org/floyd/REDparameters.txt, November, 1997.

[12] 김종현, "병렬컴퓨터구조론", 생능출판사, 1996.

[13] Robert Morris and D. Lin, "Variance of Aggregated Web Traffic", Infocom, 2000.

[14] J. R. Parker, "Genetic Algorithms and Evolutionary Computing", CPCS501 notes, 2002.

◐ 저 자 소 개 ◑

송 자 영

1998년 호서대학교 전자계산학과 졸업(이학사)
2000년 동국대학교 대학원 컴퓨터공학과 졸업(공학석사)
2009년 동국대학교 영상정보통신대학원 네트워크관리학과 졸업(공학박사)
1999~현재 동구여자상업고등학교 교사
관심분야 : 분산운영체제, 라우터 혼잡제어, 고성능 컴퓨팅, 리눅스 커널 최적화 etc.
E-mail : hpc3341@daum.net

최 병 석

1985년 서울대학교 전자공학과 졸업(공학사)
1987년 미국 Fairleigh Dickinson University EE 졸업(공학석사)
1993년 미국 Polytechnic University EE 졸업(공학석사)
1994년 미국 Polytechnic University EE 졸업(공학박사)
1997~현재 동국대학교 정보통신공학과 교수
관심분야 : 초고속 위성망, 초고속 통신망의 트래픽제어, 광대역 접속 방식, 라우터 etc.
E-mail : bchoe@dgu.edu